James Foxall

STARTER KIT
DVD includes
Visual C#® 2008
Express Edition

Sams Teach Yourself

Visual C#® 2008

in 24 Hours

SAMS

Sams Teach Yourself Visual C#® 2008 in 24 Hours: Complete Starter Kit

ISBN-13: 978-0-672-32990-6
ISBN-10: 0-672-32990-5

Library of Congress Cataloging-in-Publication data is on file

Printed in the United States of America

First Printing June 2008

Trademarks

Warning and Disclaimer

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside the U.S., please contact

International Sales
international@pearson.com

Editor-in-Chief
Karen Gettman

Executive Editor
Neil Rowe

Development Editor
Mark Renfrow

Managing Editor
Patrick Kanouse

Senior Project Editor
Tonya Simpson

Copy Editor
Margo Catts

Indexer
Tim Wright

Proofreader
Kathy Ruiz

Technical Editor
Todd Meister

Publishing Coordinator
Cindy Teeters

Multimedia Developer
Dan Scherf

Book Designer
Gary Adair

Introduction

With Microsoft's introduction of the .NET platform, a new, exciting programming language was born. Visual C# is now the language of choice for developing on the .NET platform, and Microsoft has even written a majority of the .NET Framework using Visual C#. Visual C# is a modern object-oriented language designed and developed from the ground up with a best-of-breed mentality, implementing and expanding on the best features and functions found in other languages. Visual C# 2008 combines the power and flexibility of C++ with some of the simplicity of Visual C#.

Audience and Organization

This book is targeted toward those who have little or no programming experience or who might be picking up Visual C# as a second language. The book has been structured and written with a purpose: to get you productive as quickly as possible. I've used my experiences in writing applications with Visual C# and teaching Visual C# to create a book that I hope cuts through the fluff and teaches you what you need to know. All too often, authors fall into the trap of focusing on the technology rather than on the practical application of the technology. I've worked hard to keep this book focused on teaching you practical skills that you can apply immediately toward a development project. Feel free to post your suggestions or success stories at www.jamesfoxall.com/forums.

This book is divided into five parts, each of which focuses on a different aspect of developing applications with Visual C# 2008. These parts generally follow the flow of tasks you'll perform as you begin creating your own programs with Visual C# 2008. I recommend that you read them in the order in which they appear.

- Part I, "The Visual C# 2008 Environment," teaches you about the Visual C# environment, including how to navigate and access Visual C#'s numerous tools. In addition, you'll learn about some key development concepts such as objects, collections, and events.
- Part II, "Building a User Interface," shows you how to build attractive and functional user interfaces. In this part, you'll learn about forms and controls—the user interface elements such as text boxes and list boxes.
- Part III, "Making Things Happen: Programming," teaches you the nuts and bolts of Visual C# 2008 programming—and there's a lot to learn. You'll discover how to create classes and procedures, as well as how to store data, perform loops, and make

decisions in code. After you've learned the core programming skills, you'll move into object-oriented programming and debugging applications.

- Part IV, "Working with Data," introduces you to working with graphics, text files, and programming databases, and shows you how to automate external applications such as Word and Excel. In addition, this part teaches you how to manipulate a user's file system and the Windows Registry.
- Part V, "Deploying Solutions and Beyond," shows you how to distribute an application that you've created to an end user's computer. In Hour 24, "The 10,000-Foot View," you'll learn about Microsoft's .NET initiative from a higher, less-technical level.

Many readers of previous editions have taken the time to give me input on how to make this book better. Overwhelmingly, I was asked to have examples that build on the examples in the previous chapters. In this book, I have done that as much as possible. Now, instead of learning concepts in isolated bits, you'll be building a feature-rich Picture Viewer program throughout the course of this book. You'll begin by building the basic application. As you progress through the chapters, you'll add menus and toolbars to the program, build an Options dialog box, modify the program to use the Windows Registry and a text file, and even build a setup program to distribute the application to other users. I hope you find this approach beneficial in that it enables you to learn the material in the context of building a real program.

Conventions Used in This Book

This book uses several design elements and conventions to help you prioritize and reference the information it contains:

By the Way

By the Way boxes provide useful sidebar information that you can read immediately or circle back to without losing the flow of the topic at hand.

Did you know?

Did You Know? boxes highlight information that can make your Visual C# programming more effective.

Watch Out!

Watch Out! boxes focus your attention on problems or side effects that can occur in specific situations.

New terms appear *italic* for emphasis.

In addition, this book uses various typefaces to help you distinguish code from regular English. Code is presented in a `monospace` font. Placeholders—words or characters that represent the real words or characters you would type in code—appear in *`italic monospace`*. When you are asked to type or enter text, that text appears in **bold**.

Some code statements presented in this book are too long to appear on a single line. In these cases, a line-continuation character (an underscore) is used to indicate that the following line is a continuation of the current statement.

Onward and Upward!

This is an exciting time to be learning how to program. It's my sincerest wish that when you finish this book, you feel capable of creating, debugging, and deploying modest Visual C# programs, using many of Visual C#'s tools. Although you won't be an expert, you'll be surprised at how much you've learned. And I hope this book will help you determine your future direction as you proceed down the road to Visual C# mastery.

HOUR 1

Jumping In with Both Feet: A Visual C# 2008 Programming Tour

What You'll Learn in This Hour:

- Building a simple (yet functional) Visual C# application
- Letting a user browse a hard drive
- Displaying a picture from a file on disk
- Getting familiar with some programming lingo
- Learning about the Visual Studio .NET IDE

Learning a new programming language can be intimidating. If you've never programmed before, the act of typing seemingly cryptic text to produce sleek and powerful applications probably seems like a black art, and you might wonder how you'll ever learn everything you need to know. The answer is, of course, one step at a time. The first step to learning a language is the same as that of any other activity: *building confidence*. Programming is part art and part science. Although it might seem like magic, it's more akin to illusion: After you know how things work a lot of the mysticism goes away, freeing you to focus on the mechanics necessary to produce any given desired result.

Producing large, commercial solutions is accomplished by way of a series of small steps. After you've finished creating the project in this hour, you'll have a feel for the overall development process and will have taken the first step toward becoming an accomplished programmer. In fact, you will be building upon this Picture Viewer program in subsequent chapters. By the time you complete this book, you will have built a distributable application, complete with resizable screens, an intuitive interface including menus and toolbars, and robust code with professional error handling. But I'm getting ahead of myself!

In this hour, you'll complete a quick tour that takes you step by step through creating a complete, albeit small, Visual C# program. Most introductory programming books start out with the reader creating a simple Hello World program. I've yet to see a Hello World program that's the least bit helpful (they usually do nothing more than print `hello world` to the screen—oh, what fun). So, instead, you'll create a picture viewer application that lets you view Windows bitmaps and icons on your computer. You'll learn how to let a user browse for a file and how to display a selected picture file on the screen. The techniques you learn in this chapter will come in handy in many real-world applications that you'll create, but the goal of this chapter is for you to realize just how much fun it is to program with Visual C#.

Starting Visual C# 2008

Before you begin creating programs in Visual C# 2008, you should be familiar with the following terms:

- **Distributable component**—The final, compiled version of a project. Components can be distributed to other people and other computers, and they don't require the Visual C# 2008 development environment (the tools you use to create a .NET program) to run (although they do require the .NET runtime, which I discuss in Hour 23, "Deploying Applications"). Distributable components are often called *programs*. In Hour 23, you'll learn how to distribute the Picture Viewer program that you're about to build to other computers.
- **Project**—A collection of files that can be compiled to create a distributable component (program). There are many types of projects, and complex applications might consist of multiple projects, such as a Windows application project, and support dynamic link library (DLL) projects.
- **Solution**—A collection of projects and files that make up an application or component.

By the Way

Visual C# is part of a larger entity known as the *.NET Framework*. The .NET Framework encompasses all the .NET technology, including Visual Studio .NET (the suite of development tools) and the Common Language Runtime (CLR), which is the set of files that make up the core of all .NET applications. You'll learn about these items in more detail as you progress through this book. For now, realize that Visual C# is one of many languages that exist within the .NET family. Many other languages, such as Visual Basic, are also .NET languages, make use of the CLR, and are developed within Visual Studio .NET.

Visual Studio 2008 is a complete development environment, and it's called the *IDE* (short for *integrated development environment*). The IDE is the design framework in which you build applications; every tool you'll need to create your Visual C# projects is accessed from within the Visual C# IDE. Again, Visual Studio 2008 supports development in many different languages—Visual C# being one of the most popular. The environment itself is not Visual C#, but the language you use within Visual Studio 2008 *is* Visual C#. To work with Visual C# projects, you first start the Visual Studio 2008 IDE.

Start Visual Studio 2008 now by choosing Microsoft Visual C# 2008 Express Edition on your Start/Programs menu. If you are running the full retail version of .NET, your shortcut may have a different name. In this case, locate the shortcut on your Start menu and click it once to start the Visual Studio .NET IDE.

Creating a New Project

When you first start Visual Studio .NET, you're shown the Start Page tab within the IDE. You can open projects created previously or create new projects from this Start page (see Figure 1.1). For this quick tour, you're going to create a new Windows application, so open the File menu and click New Project to display the New Project dialog box shown in Figure 1.2.

By the Way

If your Start page doesn't look like the one in Figure 1.1, chances are that you've changed the default settings. In Hour 2, "Navigating Visual C# 2008," I'll show you how to change them back.

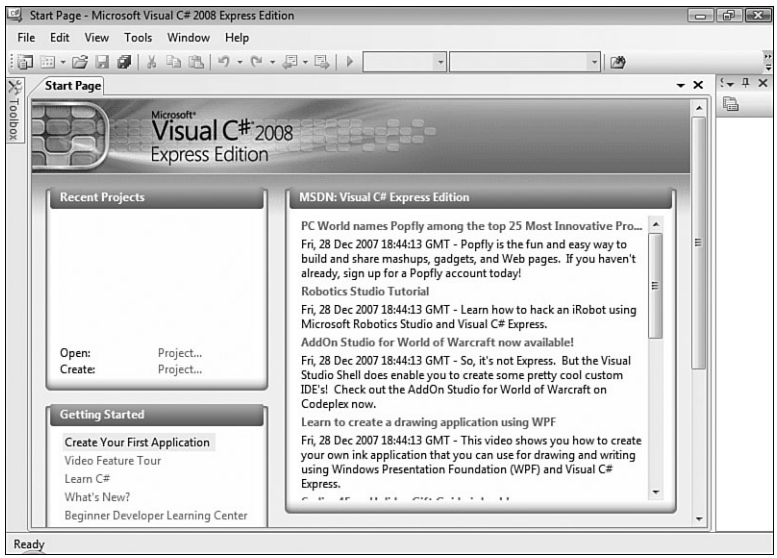

FIGURE 1.1 You can open existing projects or create new projects from the Visual Studio Start page.

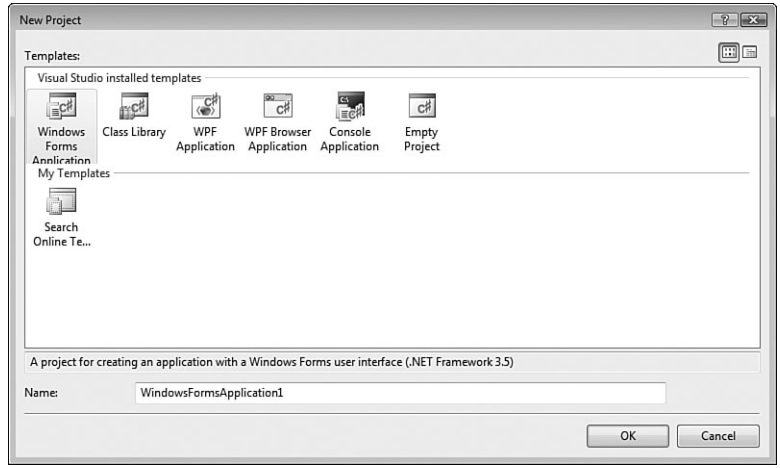

FIGURE 1.2
The New Project dialog box enables you to create many types of .NET projects.

The New Project dialog box is used to specify the type of Visual C# project to create. (You can create many types of projects with Visual C#, as well as with the other supported languages of the .NET Framework.) The options shown in Figure 1.2 are limited because I am running the Express edition of Visual C# for all examples in this books. If you are running the full version of Visual C#, many more options are available to you.

Create a new Windows application by following these steps:

1. Make sure that the Windows Application icon is selected (if it's not, click it once to select it).
2. At the bottom of the New Project dialog box is a Name text box. This is where, oddly enough, you specify the name of the project you're creating. Enter Picture Viewer in the Name text box.
3. Click OK to create the project.

Did you Know?

Always set the Name text box to something meaningful before creating a project, or you'll have more work to do later if you want to move or rename the project.

When Visual C# creates a new Windows application project, it adds one form (the empty gray window) for you to begin building the *interface*—the graphical windows with which you interact—for your application (see Figure 1.3).

Within Visual Studio 2008, form is the term given to the design-time view of windows that can be displayed to a user.

By the Way

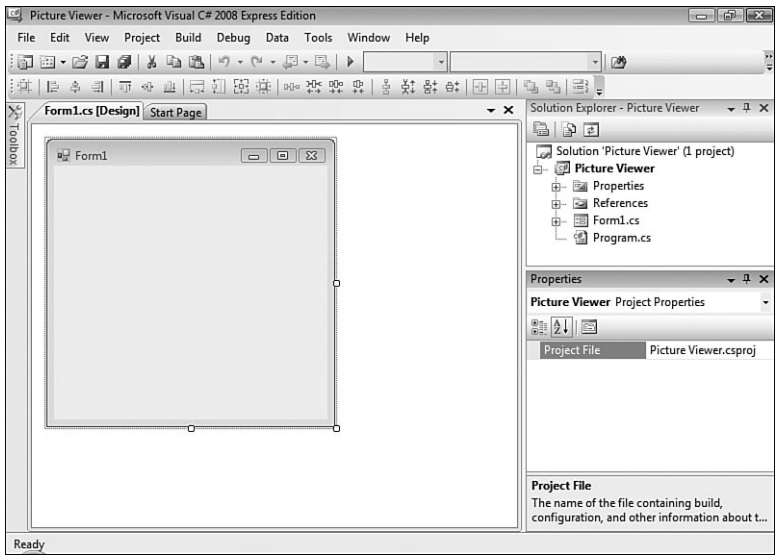

FIGURE 1.3
New Windows applications start with a blank form; the fun is just beginning!

Your Visual Studio 2008 environment might look different from that shown in the figures of this hour because of the edition of Visual Studio 2008 you're using, whether you've already played with Visual Studio 2008, and other factors such as the resolution of your monitor. All the elements discussed in this hour exist in all editions of Visual Studio 2008, however. (If a window shown in a figure isn't displayed in your IDE, use the View menu to display it.)

To create a program that can be run on another computer, you start by creating a project and then compiling the project into a component such as an *executable* (a program a user can run) or a DLL (a component that can be used by other programs and components). The compilation process is discussed in detail in Hour 23, "Deploying Applications." The important thing to note at this time is that when you hear someone refer to *creating* or *writing* a program, just as you're creating the Picture Viewer program now, they're referring to the completion of all steps up to and including compiling the project to a distributable file.

By the Way

Understanding the Visual Studio .NET Environment

The first time you run Visual Studio 2008, you'll notice that the IDE contains a number of windows, such as the Solutions Explorer window on the right, which is used to view the files that make up a project. In addition to these windows, the IDE contains a number of tabs, such as the vertical Toolbox tab on the left edge of the IDE (refer to Figure 1.3). Try this now: Click the Toolbox tab to display the Toolbox window (clicking a tab displays an associated window). You can hover the mouse over a tab for a few seconds to display the window as well. To hide the window, simply move the mouse off the window (if you hovered over the tab to display it) or click on another window. To close the window completely, click the Close (X) button in the window's title bar.

By the Way

If you opened the toolbox by clicking its tab rather than hovering over the tab, the toolbox does not automatically close. Instead, it stays open until you click on another window.

You can adjust the size and position of any of these windows, and you can even hide and show them as needed. You'll learn how to customize your design environment in Hour 2.

Watch Out!

Unless specifically instructed to do so, don't double-click anything in the Visual Studio 2008 design environment. Double-clicking most objects produces an entirely different result than single-clicking does. If you mistakenly double-click an object on a form (discussed shortly), a code window is displayed. At the top of the code window is a set of tabs: one for the form design and one for the code. Click the tab for the form design to hide the code window and return to the form.

The Properties window at the right side of the design environment is perhaps the most important window in the IDE, and it's the one you'll use most often. If your computer display resolution is set to 800×600, you can probably see only a few properties at this time. This makes it difficult to view and set properties as you create projects. All the screen shots in this book are taken at 800×600 due to size constraints, but you should run at a higher resolution if you can. I highly recommend that you develop applications with Visual C# at a screen resolution of 1024×768 or higher because it offers plenty of work space. Keep in mind, however, that end users might be running at a lower resolution than you are using for development. If you need to change your display settings, right-click your desktop and select Personalize.

Changing the Characteristics of Objects

Almost everything you work with in Visual C# is an object. Forms, for instance, are objects, as are all the items you can put on a form to build an interface such as list boxes and buttons. There are *many* types of objects, and objects are classified by type. For example, a form is a Form object, whereas items you can place on a form are called Control objects, or controls. (Hour 3, "Understanding Objects and Collections," discusses objects in detail.) Some objects don't have a physical appearance but exist only in code, and you'll learn about these kinds of objects in later hours.

Watch Out!

You'll find that I often mention material coming up in future chapters. In the publishing field, we call these *forward references*. For some reason, these tend to really unnerve some people. I do this only so that you realize you don't have to fully grasp a subject when it's first presented; the material is covered in more detail later. I try to keep forward references to a minimum, but teaching programming is, unfortunately, not a perfectly linear process. There will be times I'll have to touch on a subject that I feel you're not ready to dive into fully yet. When this happens, I give you a forward reference to let you know that the subject is covered in greater detail later on.

Every object has a distinct set of attributes known as *properties* (regardless of whether the object has a physical appearance). You have certain properties about you, such as your height and hair color. Visual C# objects have properties as well, such as Height and BackColor. Properties define an object's characteristics. When you create a new object, the first thing you need to do is set its properties so that the object appears and behaves the way you want it to. To display an object's properties, click the object in its designer (the main work area in the IDE).

First, make sure your Properties Window is displayed by opening the View menu and choosing Properties Window. Next, click anywhere in the default form now (its title bar says Form1) and check to see whether its properties are displayed in the Properties window. You'll know because the drop-down list box at the top of the properties window contains the form's name: Form1 System.Windows.Forms.Form. Form1 is the name of the object, and System.Windows.Forms.Form is the type of object.

Naming Objects

The property you should always set first for any new object is the Name property. Scroll toward the top of the properties list until you see the `(Name)` property (see Figure 1.4). If the `Name` property isn't one of the first properties listed, your properties

window is set to show properties categorically instead of alphabetically. You can show the list alphabetically by clicking the Alphabetical button that appears just above the properties grid.

By the Way

I recommend that you keep the Properties window set to show properties in alphabetical order; doing so makes it easier to find properties that I refer to in the text. Note that the `Name` property always stays toward the top of the list and is referred to as `(Name)`. If you're wondering why it has parentheses around it, that's because the parentheses force the property to the top of the list because symbols come before letters in an alphabetical sort.

When saving a project, you choose a name and a location for the project and its files. When you first create an object, Visual C# gives the object a unique, generic name based on the object's type. Although these names are functional, they simply aren't descriptive enough for practical use. For instance, Visual C# named your form Form1, but it's common to have dozens of forms in a project, and it would be extremely difficult to manage such a project if all forms were distinguishable only by a number (Form2, Form3, and so forth).

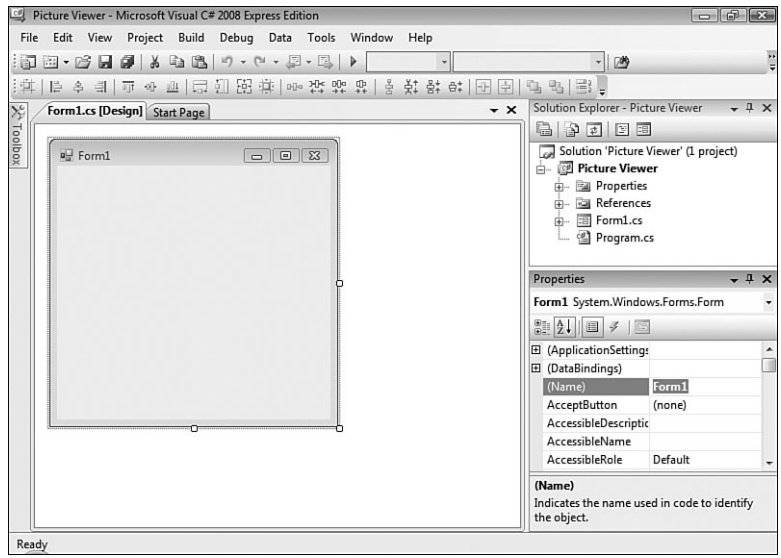

FIGURE 1.4
The `Name` property is the first property you should change when you add a new object to your project.

By the Way

What you're actually working with is a *form class*, or *template*, that will be used to create and show forms at runtime. For the purpose of this quick tour, I simply refer to it as a form. See Hour 5, "Building Forms—The Basics," for more information.

To better manage your forms, give each one a descriptive name. Visual C# gives you the chance to name new forms as they're created in a project. Visual C# created this default form for you, so you didn't get a chance to name it. It's important to not only change the form's name but also to change its filename. Change the programmable name and the filename at the same time by following these steps:

1. Click the `Name` property and change the text from Form1 to ViewerForm. Notice that this does not change the form's filename as it's displayed in the Solution Explorer window located above the Properties window.
2. Right-click `Form1.cs` in the Solution Explorer window (the window above the properties window).
3. Choose Rename from the context menu that appears.
4. Change the text from Form1.cs to **ViewerForm.cs**.

By the Way

I use the Form suffix here to denote that the file is a form class. Suffixes are optional, but I find they really help you keep things organized.

The Name property of the form is actually changed for you automatically when you rename the file. I had you explicitly change the Name property because it's something you're going to be doing a lot—for all sorts of objects.

Setting the `Text` Property of the Form

Notice that the text that appears in the form's title bar says Form1. Visual C# sets the form's title bar to the name of the form when it's first created but doesn't change it when you change the form's name. The text in the title bar is determined by the value of the `Text` property of the form. Change the text now by following these steps:

1. Click the form once more so that its properties appear in the Properties window.
2. Use the scrollbar in the Properties window to locate the `Text` property.
3. Change the text to **Picture Viewer**. Press the Enter key or click on a different property. You'll see the text in the title bar of the form change.

Saving a Project

The changes you've made so far exist only in memory; if you were to turn off your computer at this time, you would lose all your work up to this point. Get into the habit of frequently saving your work, which commits your changes to disk.

Click the Save All button on the toolbar (the picture of a stack of disks) now to save your work. Visual C# then displays the Save Project dialog box shown in Figure 1.5. Notice that the `Name` property is already filled in because you named the project when you created it. The Location text box is where you specify the location in which to save the project. Visual C# creates a subfolder in this location, using the value in the Name text box (in this case, Picture Viewer). You can use the default location, or change it to suit your purposes. You can have Visual C# create a solution folder in which the project folder gets placed. On large projects, this is a handy feature. For now, it's an unnecessary step, so uncheck the Create Directory for Solution box and then click Save to save the project.

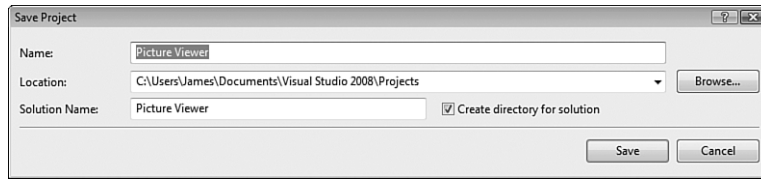

FIGURE 1.5
When saving a project, choose a name and a location for the project and its files.

Giving the Form an Icon

Everyone who has used Windows is familiar with icons—the little pictures that represent programs. Icons most commonly appear in the Start menu next to the names of their respective programs. In Visual C#, you not only have control over the icon of your program file, you can also give every form in your program a unique icon if you want to.

By the Way

The following instructions assume that you have access to the source files for the examples in this book. They are available at www.samspublishing.com. You can also get these files, as well as discuss this book, at my website at http://www.jamesfoxall.com/books.aspx. When you unzip the samples, a folder is created for each hour, and within each hour's folder are subfolders for the sample projects. You can find the icon in the folder Hour 1\Picture Viewer.

You don't have to use the icon I've provided for this example; you can use any icon of your choice. If you don't have an icon available (or you want to be a rebel), you can skip this section without affecting the outcome of the example.

To give the form an icon, follow these steps:

1. In the Properties window, click the `Icon` property to select it.
2. When you click the `Icon` property, a small button with three dots appears to the right of the property. Click this button.

3. Use the Open dialog box that appears to locate the `PictureViewer.ico` file or another icon file of your choice. When you've found the icon, double-click it, or click it once to select it and then click Open.

After you've selected the icon, it appears in the `Icon` property along with the word "Icon." A small version of the icon appears in the upper-left corner of the form as well. Whenever this form is minimized, this is the icon displayed on the Windows taskbar.

Changing the Size of the Form

Next, you're going to change the `Width` and `Height` properties of the form. The `Width` and `Height` values are shown collectively under the `Size` property; `Width` appears to the left of the comma, `Height` to the right. You can change the `Width` or `Height` property by changing the corresponding number in the `Size` property. Both values are represented in pixels (that is, a form that has a `Size` property of 200,350 is 200 pixels wide and 350 pixels tall). To display and adjust the `Width` and `Height` properties separately, click the small plus sign (+) next to the `Size` property (see Figure 1.6).

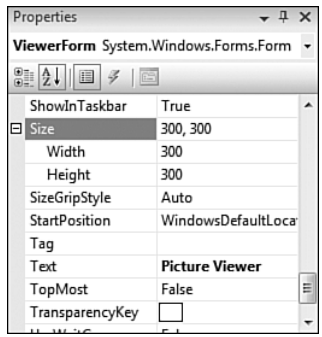

FIGURE 1.6 Some properties can be expanded to show more specific properties.

By the Way

A pixel is a unit of measurement for computer displays; it's the smallest visible "dot" on the screen. The resolution of a display is always given in pixels, such as 800×600 or 1024×768. When you increase or decrease a property by one pixel, you're making the smallest possible visible change to the property.

Change the `Width` property to 400 and the `Height` to 325 by typing in the corresponding box next to a property name. To commit a property change, press Tab or Enter, or click a different property or window. Your screen should now look like the one in Figure 1.7.

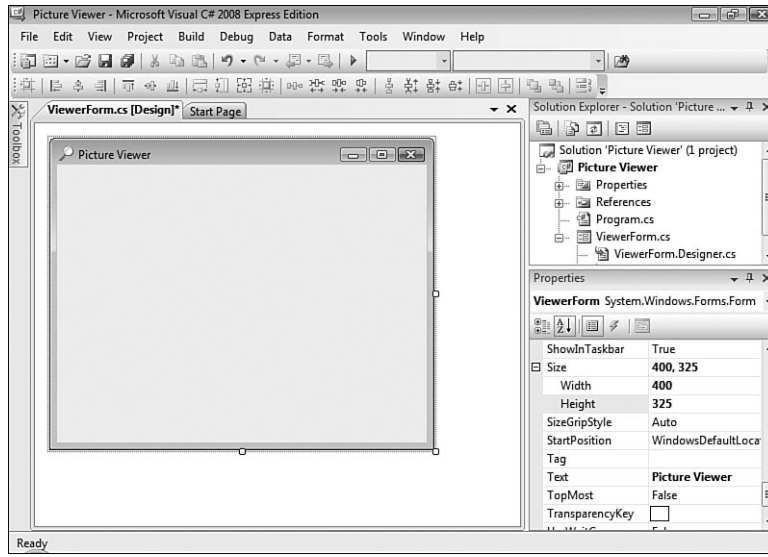

FIGURE 1.7 Changes made in the Properties window are reflected as soon as they're committed.

By the Way

> You can also size a form by dragging its border, which you'll learn about in Hour 2. This property can also be changed by program code, which you'll learn how to write in Hour 5.

Save the project now by choosing File, Save All from the menu or by clicking the Save All button on the toolbar—it has a picture of stacked disks on it.

Adding Controls to a Form

Now that you've set your form's initial properties, it's time to create a user interface by adding objects to the form. Objects that can be placed on a form are called *controls*. Some controls have a visible interface with which a user can interact, whereas others are always invisible to the user. You'll use controls of both types in this example. On the left side of the screen is a vertical tab titled Toolbox. Click the Toolbox tab now to display the Toolbox window and click the plus sign next to Common Controls to see the most commonly used controls (see Figure 1.8). The toolbox contains all the controls available in the project, such as labels and text boxes.

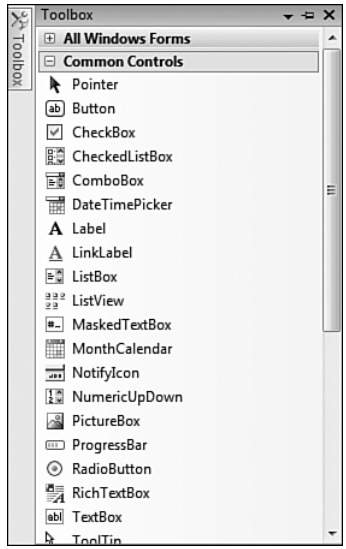

FIGURE 1.8
The toolbox is used to select controls to build a user interface.

The toolbox closes as soon as you've added a control to a form and when the pointer is no longer over the toolbox. To make the toolbox stay visible, click the little picture of a pushpin located in the toolbox's title bar.

I don't want you to add them yet, but your Picture Viewer interface will consist of the following controls:

- **Two `Button` controls**—The standard buttons that you're used to clicking in pretty much every Windows program you've ever run
- **A `PictureBox` control**—A control used to display images to a user
- **An `OpenFileDialog` control**—A hidden control that exposes the Windows Open File dialog box functionality

Designing an Interface

It's generally best to design a form's user interface and then add the code behind the interface to make the form functional. You'll build your interface in the following sections.

Adding a Visible Control to a Form

Start by adding a `Button` control to the form. Do this by double-clicking the Button item in the toolbox. Visual C# creates a new button and places it in the upper-left corner of the form (see Figure 1.9).

FIGURE 1.9 When you double-click a control in the toolbox, the control is added to the upper-left corner of the form.

Using the Properties window, set the button's properties as follows. Remember, when you view the properties alphabetically, the `Name` property is listed first, so don't go looking for it down in the list or you'll be looking awhile.

Property	Value
`Name`	`btnSelectPicture`
`Location`	`295,10` (*Note: 295 is the x coordinate; 10 is the y coordinate.*)
`Size`	`85, 23`
`Text`	`Select Picture`

You're now going to create a button that the user can click to close the Picture Viewer program. Although you could add another new button to the form by double-clicking the `Button` control on the toolbox again, this time you'll add a button to the form by creating a copy of the button you've already defined. This enables you to easily create a button that maintains the size and other style attributes of the original button when the copy was made.

To do this, right-click the Select Picture button and choose Copy from its shortcut menu. Next, right-click anywhere on the form and choose Paste from the form's shortcut menu (you could have also used the keyboard shortcuts Ctrl+C to copy and Ctrl+V to paste). The new button appears centered on the form, and it's selected by

default. Notice that it retained almost all of the properties of the original button, but the name has been reset. Change the new button's properties as follows:

Property	Value
Name	btnQuit
Location	295, 40
Text	Quit

The last visible control you need to add to the form is a `PictureBox` control. A `PictureBox` has many capabilities, but its primary purpose is to show pictures, which is precisely what you'll use it for in this example. Add a new `PictureBox` control to the form by double-clicking the PictureBox item in the toolbox and set its properties as follows:

Property	Value
Name	picShowPicture
BorderStyle	FixedSingle
Location	8, 8
Size	282, 275

After you've made these property changes, your form will look like the one in Figure 1.10. Click the Save All button on the toolbar to save your work.

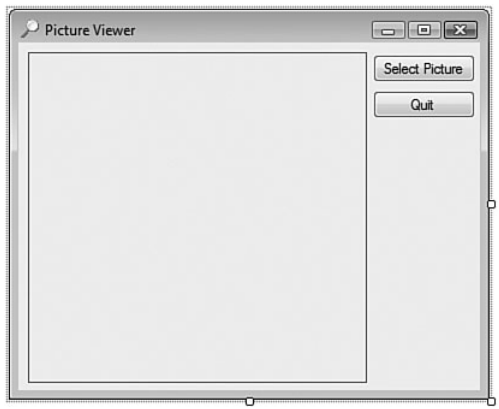

FIGURE 1.10
An application's interface doesn't have to be complex to be useful.

Adding an Invisible Control to a Form

All the controls that you've used so far sit on a form and have a physical appearance when the application is run by a user. Not all controls have a physical appearance,

however. Such controls, referred to as *nonvisual controls* (or *invisible-at-runtime* controls), aren't designed for direct user interactivity. Instead, they're designed to give you, the programmer, functionality beyond the standard features of Visual C#.

To enable the user to select a picture to display, you need to make it possible to locate a file on a hard drive. You might have noticed that whenever you choose to open a file from within any Windows application, the dialog box displayed is almost always the same. It doesn't make sense to force every developer to write the code necessary to perform standard file operations, so Microsoft has exposed the functionality via a control that you can use in your projects. This control is called the `OpenFileDialog` control, and it will save you dozens and dozens of hours that would otherwise be necessary to duplicate this common functionality.

By the Way

Other controls in addition to the `OpenFileDialog` control give you file functionality. For example, the `SaveFileDialog` control provides features for enabling the user to specify a filename and path for saving a file.

Display the toolbox now and scroll down (using the down arrow in the lower part of the toolbox) until you can see the `OpenFileDialog` control (it's in the Dialogs category), and then double-click it to add it to your form. Note that the control isn't placed on the form, but rather it appears in a special area below the form (see Figure 1.11). This happens because the `OpenFileDialog` control has no form interface to display to a user. It does have an interface (a dialog box) that you can display as necessary, but it has nothing to display directly on a form.

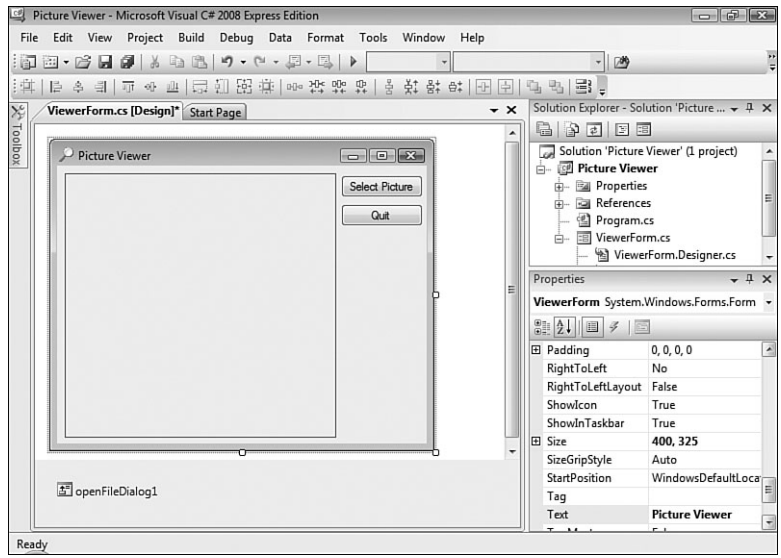

FIGURE 1.11 Controls that have no interface appear below the form designer.

Select the `OpenFileDialog` control and change its properties as follows:

Property	Value
`Name`	`ofdSelectPicture`
`Filename`	*<make empty>*
`Filter`	Windows Bitmaps|*.BMP|JPEG Files|*.JPG
`Title`	`Select Picture`

By the Way

Don't actually enter the text **<make empty>** for the filename; I really mean delete the default value and make this property value empty.

The `Filter` property is used to limit the types of files that will be displayed in the Open File dialog box. The format for a filter is *description|filter*. The text that appears before the first pipe symbol is the descriptive text of the file type, whereas the text after the pipe symbol is the pattern to use to filter files. You can specify more than one filter type by separating each description|filter value with another pipe symbol. Text entered into the `Title` property appears in the title bar of the Open File dialog box.

The graphical interface for your Picture Viewer program is now finished. If you pinned the toolbox open, click the pushpin in the title bar of the toolbox now to close it.

Writing the Code Behind an Interface

You have to write code for the program to be capable of performing tasks and responding to user interaction. Visual C# is an *event-driven* language, which means that code is executed in response to events. These events might come from users, such as a user clicking a button and triggering its `Click` event, or from Windows itself (see Hour 4, "Understanding Events," for a complete explanation of events). Currently, your application looks nice but it won't do a darn thing. Users can click the Select Picture button until they can file for disability with carpel tunnel syndrome, but nothing will happen because you haven't told the program what to do when the user clicks the button. You can see this for yourself now by pressing F5 to run the project. Feel free to click the buttons, but they don't do anything. When you're finished, close the window you created to return to Design mode.

You're going to write code to accomplish two tasks. First, you're going to write code that lets users browse their hard drives to locate and select a picture file and then display the file in the picture box (this sounds a lot harder than it is). Second, you're

going to add code to the Quit button that shuts down the program when the user clicks the button.

Letting a User Browse for a File

The first bit of code you're going to write enables users to browse their hard drives, select a picture file, and then see the selected picture in the `PictureBox` control. This code executes when the user clicks the Select Picture button; therefore, it's added to the `Click` event of that button.

When you double-click a control on a form in Design view, the default event for that control is displayed in a code window. The default event for a `Button` control is its `Click` event, which makes sense because clicking is the most common action a user performs with a button. Double-click the Select Picture button now to access its `Click` event in the code window (see Figure 1.12).

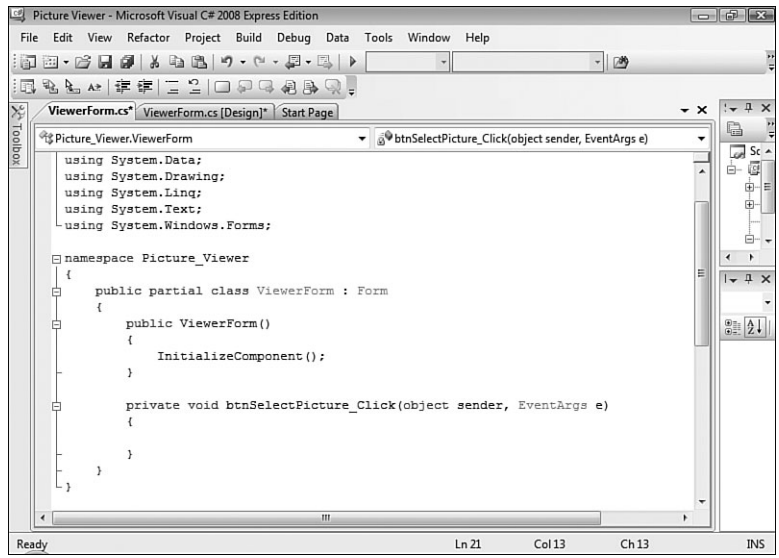

FIGURE 1.12 You'll write all code in a window such as this.

When you access an event, Visual C# builds an `event handler`, which is essentially a template procedure in which you add the code that executes when the event occurs. The cursor is already placed within the code procedure, so all you have to do is add code. Although this may seem daunting to you now, by the time you're finished with this book you'll be madly clicking and clacking away as you write your own code to make your applications do exactly what you want them to do—well, most of the time. For now, just enter the code as I present it here.

It's important that you get in the habit of commenting your code, so the first statement you're going to enter is a comment. Beginning a statement with two forward slashes designates the statement as a comment; the compiler doesn't do anything with the statement, so you can enter whatever text you want after the two forward slashes. Type the following statement exactly as it appears and press the Enter key at the end of the line:

```
// Show the open file dialog box.
```

The next statement you enter triggers a method of the `OpenFileDialog` control that you added to the form. You'll learn all about methods in Hour 3. For now, think of a method as a mechanism to make a control do something. The `ShowDialog()` method tells the control to show its Open dialog box and let the user select a file. The `ShowDialog()` method returns a value that indicates its success or failure, which you'll then compare to a predefined result (`DialogResult.OK`). Don't worry too much about what's happening here; you'll be learning the details of all this in later hours, and the sole purpose of this hour is to get your feet wet. In a nutshell, the `ShowDialog()` method is invoked to let a user browse for a file. If the user selects a file, more code is executed. Of course, there's a lot more to using the `OpenFileDialog` control than I present in this basic example, but this simple statement gets the job done. Enter the following two code statements, pressing Enter at the end of each line:

By the Way

Capitalization is important. Visual C# is a *case-sensitive language*, which means `ShowDialog()` is not the same as `Showdialog()`. If you get the case of even one letter wrong, Visual C# doesn't recognize the word and your code doesn't work, so always enter code exactly as it appears in this book!

```
if (ofdSelectPicture.ShowDialog() == DialogResult.OK)
{
```

The opening brace (the { character) is necessary for this `if` statement because it denotes that this `if` construct will be made up of multiple lines.

Time for another comment. Your cursor is currently on the line below the { that you entered. Type this statement and remember to press Enter at the end of the code line.

```
// Load the picture into the picture box.
```

This next statement is the line of code that actually displays the picture in the picture box.

Enter the following statement:

```
picShowPicture.Image = Image.FromFile(ofdSelectPicture.FileName);
```

In addition to displaying the selected picture, your program is also going to display the path and filename of the picture in the title bar. When you first created the form, you changed the form's `Text` property in the Properties window. To create dynamic applications, properties need to be constantly adjusted at runtime, and this is done with code. Insert the following two statements (press Enter at the end of each line):

```
// Show the name of the file in the form's caption.
this.Text = string.Concat("Picture Viewer(" + ofdSelectPicture.FileName + ")");
```

The last statement you need to enter is a closing brace (a } character). Whenever you have an opening brace, you have to have a closing brace. This is how Visual C# groups multiple statements of code. Enter this statement now:

```
}
```

After you've entered all the code, your editor should look like that shown in Figure 1.13.

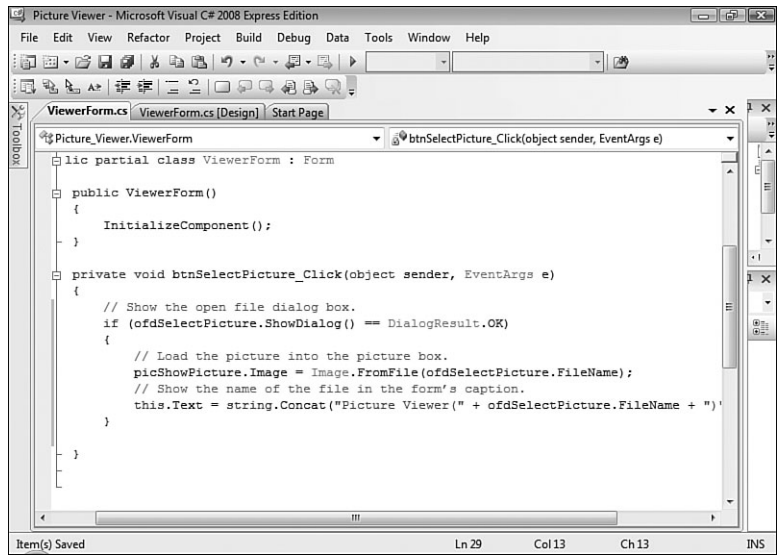

FIGURE 1.13 Make sure that your code exactly matches the visible code shown here.

Terminating a Program Using Code

The last bit of code you'll write terminates the application when the user clicks the Quit button. To do this, you'll need to access the `Click` event handler of the `btnQuit` button. At the top of the code window are two tabs. The current tab has the text ViewerForm.cs*. This is the tab containing the code window for the form with the filename `ViewerForm.cs`. Next to this is a tab that contains the text

ViewerForm.cs [Design]*. Click this tab now to switch from Code view to the form designer. If you receive an error when you click the tab, the code you entered contains an error, and you need to edit it to make it the same as shown in Figure 1.13. After the form designer appears, double-click the Quit button to access its `Click` event.

Enter the following code in the Quit button's `Click` event handler and press Enter at the end of each statement:

```
// Close the window and exit the application
this.Close();
```

By the Way

The `this.Close();` statement closes the current form. When the last loaded form in a program is closed, the application shuts itself down—completely. As you build more robust applications, you'll probably want to execute all kinds of clean-up routines before terminating an application, but for this example, closing the form is all you need to do.

Running a Project

Your application is now complete. Click the Save All button on the toolbar (it looks like a stack of disks), and then run your program by pressing F5. You can also run the program by clicking the button on the toolbar that looks like a right-facing triangle and resembles the Play button on a DVD (this button is called Start, and it can also be found on the Debug menu). Learning the keyboard shortcuts will make your development process move along faster, so I recommend you use them whenever possible.

When you run the program, the Visual C# interface changes, and the form you've designed appears floating over the design environment (see Figure 1.14).

FIGURE 1.14 When in Run mode, your program executes the same as it would for an end user.

You are now running your program as though it were a standalone application running on another user's machine; what you see is exactly what users would see if they ran the program (without the Visual Studio 2008 design environment in the background, of course). Click the Select Picture button to display the Select Picture dialog box (see Figure 1.15). Use the dialog box to locate a picture file. When you've found a file, double-click it, or click once to select it and then click Open. The selected picture is then displayed in the picture box, as shown in Figure 1.16.

By the Way

When you click the Select Picture button, the default path shown depends on the last active path in Windows, so it might be different for you than what is shown in Figure 1.15.

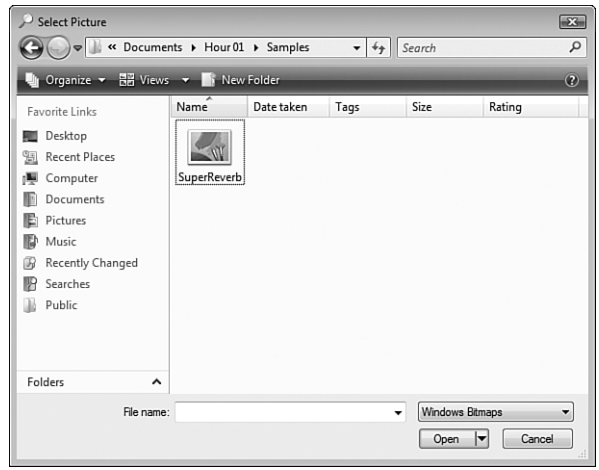

FIGURE 1.15
The `OpenFileDialog` control handles all the details of browsing for files. Cool, huh?

FIGURE 1.16
What could be prettier than a 1964 Fender Super Reverb amplifier?

By the Way

If you want to select and display a picture from your digital camera, chances are the format is JPEG, so you need to select this from the Files of Type drop-down. Also, if your image is very large, you'll see only the upper-left corner of the image (what fits in the picture box). In later hours, I'll show you how you can scale the image to fit the picture box, and even resize the form to show a larger picture in its entirety.

Summary

When you're finished playing with the program, click the Quit button to return to Design view.

That's it! You've just created a bona fide Visual C# program. You've used the toolbox to build an interface with which users can interact with your program, and you've written code in strategic event handlers to empower your program to do things. These are the basics of application development in Visual C#. This fundamental approach is used to build even the most complicated programs; you build the interface and add code to make the application do things. Of course, writing code to do things *exactly* the way you want things done is where the process can get complicated, but you're on your way.

If you take a close look at the organization of the hours in this book, you'll see that I start out by teaching you the Visual C# (Visual Studio 2008) environment. I then move on to building an interface, and later I teach you all about writing code. This organization is deliberate. You might be a little anxious to jump in and start writing serious code, but writing code is only part of the equation—don't forget the word *Visual* in Visual C#. As you progress through the hours, you'll be building a solid foundation of development skills.

Soon, you'll pay no attention to the man behind the curtain—you'll *be* that man (or woman)!

Q&A

Q. ***Can I show pictures of file types other than BMP and JPG?***

A. Yes. The `PictureBox` control supports the display of images with the extensions BMP, JPG, ICO, EMF, WMF, and GIF. The `PictureBox` control can even save images to a file using any of the supported file types.

Q. ***Is it possible to show pictures in other controls?***

A. `PictureBox` is the control to use when you are just displaying images. However, many other controls enable you to display pictures as part of the control. For instance, you can display an image on a button control by setting the button's `Image` property to a valid picture.

Workshop

The Workshop is designed to help you anticipate possible questions, review what you've learned, and get you thinking about how to put your knowledge into practice.

Quiz

1. What type of Visual C# project creates a standard Windows program?
2. What window is used to change the attributes (location, size, and so on) of a form or control in the IDE?
3. How do you access a control's default event (code)?
4. What property of a picture box do you set to display an image?
5. What is the default event for a button control?

Answers

1. Windows Forms Application
2. The Properties window
3. Double-click the control in the designer
4. The `Image` property
5. The `Click` event

Exercises

1. Change your Picture Viewer program so that the user can also locate and select GIF files. (Hint: Change the `Filter` property of the `OpenFileDialog` control.)
2. Create a new project with a new form. Create two buttons on the form, one above the other. Next, change their position so that they appear next to each other.

Index

SYMBOLS

A

B

C

D

E

F

G

H

I

J - K - L

M

N

O

P

Q - R

S

T

U - V

W

X - Y - Z